Appendix B

PhoenixPhlash NT

Flash ROM Programming Utility for Windows NT

The PhoenixAD driver referred to in Appendix B is the Access Driver 46.

1.0 Overview

The PhoenixPhlash flash utility will be used to program BIOS images into flash ROMs in AT compatible systems. The utility will consist of the following files:

PHLASHNT.EXE	- used to program the flash ROM
PLATFORM.DLL	- used to perform platform dependent functions
BIOS.ROM	- actual BIOS image to be programmed into flash ROM

This specification provides a detailed description of the functionality for the PHLASHNT.EXE program. Because PLATFORM.DLL and BIOS.ROM are platform specific, only the general format of these two files is covered in this document.

PhoenixPhlash will be executed as a Win32 console application.

High priority is being placed on flexibility, adaptability and supportability for this design project. As much customization capability as possible will be placed into the PLATFORM.DLL file so that many different platforms and configurations can be supported without modifying PHLASHNT.EXE. PhoenixPhlash will support platforms with a single flash ROM part, as well as platforms with multiple flash ROMs. Flash ROMs from 1Mbit to 4Mbits or greater will be accommodated, including boot block devices and devices with any configuration of multiple flashable regions.

For each supported part, all code specific to the particular flash ROM part (e.g. Intel 28Fxxxx) will be part of the PHLASHNT.EXE module. All code and parameters specific to a platform (e.g. flash enable code and flash ROM address range) will be part of the PLATFORM.DLL module.

PHLASHNT.EXE, the main module of the PhoenixPhlash utility, will contain all code which is platform independent. It will contain user interface code, code to load and verify the PLATFORM.DLL file and the platform independent portions of code to program a flash device.

PHLASHNT.EXE will be a Win32 executable file, generated using Microsoft C++ V4.2 or later.

Using CDriver to access hardware

PHLASHNT.EXE uses the CDriver C++ class which works in conjunction with the *PhoenixAD* driver to enable Windows NT user-mode applications to access I/O ports; to access BIOS data and code area; and to execute BIOS32 services. The CDriver class provides a simple and flexible interface between the application program and the *PhoenixAD* driver.

CDriver works in conjunction with the *PhoenixAD* driver to provide the following functions to user-mode application programs.

Access to I/O ports
Execute BIOS32 services

Access the BIOS image
Access BIOS data areas
Read the system Real Time Clock

The CDriver class serves as a thin wrapper interface between Windows NT applications and the *PhoenixAd* driver. It encapsulates the interface to the driver and provides flexibility to both applications and the kernel driver designs.

To assure future compatibility, PHLASHNT.EXE does not call the *PhoenixAD* driver directly; instead, it calls the methods in the CDriver class.

2.0 Modes of Operation

2.1 Win32 Console

PHLASHNT.EXE will be started in a Windows NT window, followed by optional command line flags (if any).

Command line flags will include (both lower and upper case characters are acceptable):

/A

AUTODETECT OFF - Do not read ID from the part. By default, program verifies that the manufacturer ID and part ID read from the part, matches the ID specified in the PLATFORM.DLL file and when the two IDs differ, the ID read from the part is used. This allows one to use the same PLATFORM.DLL and BIOS.ROM for several different parts without the need to modify either of the two files. When this flag is not set, then it is assumed that the ID is "readable" from the part. When this flag is set, the ID from the part is not used, instead the values specified in the PLATFORM.DLL are used.

/B=*filename*

BINARY FILE - Overrides the default platform specific binary file. This option is required when a full path specification is needed and/or the binary file has a name other than PLATFORM.DLL.

/BU=*filename*

BACKUP - Save the previous version of the BIOS image into file *filename* before erasing. *Filename* is optional; if not specified, the previous image is stored in BIOS.BAK. Because many versions of BIOS use platform dependent features such as shadow memory and de-compression, it is often necessary to use platform dependent code in PLATFORM.DLL to retrieve the BIOS image before it can written to a file.

/C

CMOS UPDATE - Clears the CMOS checksum after Flash is updated. If the AUTO_UPDATE feature is installed in the new BIOS image, the BIOS automatically sets all CMOS fields to their default values on the next boot. If the AUTO_UPDATE feature is not loaded, the BIOS displays the CMOS checksum error message on the next boot and prompts the user to press the F2 key to execute Setup and manually reconfigure the machine.

/CS

CHECKSUM BIOS ROM - Computes checksum on BIOS ROM image. If the checksum is not zero, or if the optional PLATFORM.DLL function CheckSum fails, the program terminates with an error message.

/H

USAGE - Display program name, version, copyright and help screen. /? can also be used for this option.

/I

IMAGE SIZE VERIFICATION - Proceed only if the ROM image file size is the same as the size of the flash part.

/MODE=n

OPERATION - Selects the operating mode for PHLASHNT. The following operating modes are currently supported:

0 Update only the BIOS image (the normal operating mode). In this mode, PHLASHNT replaces the current BIOS image with the new image. The DMI information in the system BIOS is maintained. This is the default mode and is selected if the /MODE command line flag isn't present or if an operating mode isn't specified.

1 Update only the DMI information. In this mode, PHLASHNT writes the strings specified via the DMI command line flags to the Flash. The DMI information in the system BIOS is maintained unless new DMI strings are specified on the command line.

2 Update both the BIOS and DMI information (save system DMI strings). In this mode, PHLASHNT both replaces the current BIOS image and writes the strings specified via the DMI command line flags to Flash. The DMI information in the system BIOS is maintained unless new DMI strings are specified on the command line.

3 Update both the BIOS and DMI information (reset system DMI strings). In this mode, PHLASHNT both replaces the current BIOS image and writes the strings specified via the DMI command line flags to Flash. The DMI information in the system BIOS is replaced with the DMI strings from the new BIOS ROM image and/or new DMI strings specified on the command line.

These options are not displayed by the help (/H) option for security reasons.

/N

NEW (different) - Proceed only for different version of BIOS ROM. If the data structure at BCPSYS, which includes BIOS version and build date & time, is same as the corresponding structure in the BIOS.ROM file then the programs terminates without flashing.

/O

OVERRIDE PLATFORM.DLL OPTIONS - Disable all flags set in PLATFORM.DLL. Without this switch, options set in the PLATFORM.DLL are combined with options specified on the command line. When this switch is used, only command line options are used.

/P

PRODUCTION - Maximize speed of flashing. All user feedback is reduced to a minimum (no sound, or screen update). This is used to reduce the time needed to flash a part in a production environment. Only the final success/failure is reported.

/PN
BIOS PART NUMBER CHECK - Proceed only if the BIOS part number in BIOS.ROM is the same as the part number in the current BIOS.

/PF="list of options"
Command line options to be passed to the platform dependent module PLATFORM.DLL. On some platforms it may be desirable to pass command line options to the platform dependent procedures. This is done via the CmdLine() function. When both the CmdLine() address is non-zero and this command line option is present, then the string immediately following the equal sign will be passed to PLATFORM.DLL (enclose the string in double quotes if the string includes spaces).

/Rn
RETRY - If flashing a block fails, retry n times instead of aborting.
The /Rn option can be used in crisis mode by setting psiRetryCount with the desired retry count in PLATFORM.CPP.

/S
SILENT - Silent operation without audio feedback.

/V
VERIFY - After each block is programmed, data in the flash part address space will be compared to the data in the BIOS.ROM file. Any discrepancies are reported and the program will either re-try programming of the same block or the system will halt (depending on the response to a prompt). Because the check is made after the flash memory was erased, the system will be very unstable and it may not be possible to properly notify the user and recover.

/Z
ZERO BLOCKS - Zero flash blocks before erasing.

filename
BIOS ROM image file name. Any command line option without the leading back-slash will be interpreted as the file name for the BIOS ROM image file. A filename is only required when necessary to specify a full path for the ROM BIOS image and/or the ROM BIOS image file is different than BIOS.ROM.

@filename
Response file. Any of the command line options described above may be placed in a response file. PHLASHNT will read the file and process the options as though they were entered on the command line. The options may be placed on a single line or on separate lines. Each line may be up to 1024 characters in length.

The following command line flags are used to write information to Flash for later retrieval through the Phoenix Desktop Management Interface (DMI). DMI command line flags are ignored if the target BIOS image does not support the DMI interface (doesn't have a DMI BCP structure installed) or the PHLASHNT operation mode is BIOS only (see above).

All flags have the format /D*xx*:*String*, where *xx* is one or two characters identifying the specific DMI string (see below). DMI command line flags are optional; *i.e.*, if a given DMI command line flag isn't specified, the previous contents of the corresponding DMI string buffer aren't modified, unless a default string is specified in PLATFORM.DLL. In this case, PHLASHNT always writes the default string to the corresponding DMI string buffer. If a DMI command flag is specified without the *String* field, the corresponding DMI string buffer is cleared (set to a null string). *String* can only contain printable ASCII characters. *String* must be enclosed in quotes if it contains spaces. The maximum length of each DMI string is platform specific; PHLASHNT returns an error if the passed string is longer than the corresponding target buffer. The following DMI fields are currently supported. These options are not displayed by the help (/H) option for security reasons.

/DSS:*String* Specifies the system serial number string.

/DMS:*String* Specifies the system manufacturer's name string.

/DPS:*String* Specifies the system product (model) identification string.

/DVS:*String* Specifies the system version string.

/DSM:*String* Specifies the motherboard serial number string.

/DMM:*String* Specifies the motherboard manufacturer's name string.

/DPM:*String* Specifies the motherboard product (model) identification string.

/DVM:*String* Specifies the motherboard version string.

/DSC:*String* Specifies the chassis serial number string.

/DMC:*String* Specifies the chassis manufacturer's name string.

/DPC:*String* Specifies the chassis product (model) identification string.

/DVC:*String* Specifies the chassis version string.

/DO1:*String* Specifies OEM string 1.

...

/DO*n*:*String* Specifies OEM string n.

The system and chassis switches are available only with DMI version 2.0.

The older forms of the command line switches, given below, were originally for DMI 1.2 and are kept for compatibility. These are equivalent to /DSM, /DMM, /DPM and /DVM respectively.

/DS:*String* Specifies the motherboard serial number string.

/DM:*String* Specifies the motherboard manufacturer's name string.

/DP:*String* Specifies the motherboard product (model) identification string.

/DV:*String* Specifies the motherboard version string.

Next, the flash program will load the PLATFORM.DLL file and call the platform dependent function EnableFlash() in PLATFORM.DLL to prepare the platform for flashing. PLATFORM.DLL will indicate memory region where the BIOS ROM image is to be loaded in memory and will indicate what memory regions to use for the flash device. Memory regions may be in conventional memory or in extended memory. After the ROM image is loaded in memory device programming begins.

For each block of the flash ROM to be programmed:

1) BeginFlash() is called in PLATFORM.DLL
2) The proper flash algorithm in PHLASHNT.EXE is executed.
3) EndFlash() is called in PLATFORM.DLL.

This process is repeated for each flash block specified in PLATFORM.DLL. This allows for multiple devices on a single platform, multiple blocks within a device and block dependent initialization/termination code for each block. This also allows for automatic saving and restoring of memory regions such as boot blocks.

During flashing, progress information is presented to the user:

1) If production mode is not selected, an appropriate message window will be displayed on the screen, which will include time of day, gas gauge style progress indicator and status line message.

2) Approximately once every second a short beep is sounded.

3) At the start and completion of each step an appropriate code is sent to the debug port.

Video will be updated at least once every second. The sound will be generated approximately every second. Note that in production environment progress update can be disabled.

After flashing is complete, DisableFlash() is executed from the PLATFORM.DLL file. One of two distinct sounds will be generated to indicate success or failure of the flash process. If video is available, an appropriate message window will be displayed. After a short pause system is re-booted.

2.3 Completion Codes

Although the program will proceed through many steps and will be capable of reporting to the user status at each step, only three major stages are identified by sound and keyboard LED codes. The three major stages are:

1) Read and verify PLATFORM.DLL file
2) Perform platform specific initialization
3) Program the part

If the program fails to complete any of the three major stages of the flashing process, program will use distinct sound sequence and keyboard LEDs to inform user at which stage the program failed. At the start of the program CAPS_LOCK, NUM_LOCK and SCROLL_LOCK LEDs on the keyboard will be turned on. The failure at each of the three stages is indicated as follows:

Sound	Keyboard LEDs ON	Description
low buzz + 3 short tones	CAPS, NUM, SCROLL	1. Before reading platform.dll
low buzz + 2 short tones	CAPS, NUM	2. Before platform init
low buzz + 1 short tone	NUM	3. After platform init
1 long tone	none	Successful completion

Stage 1 - Failure occurred before locating PLATFORM.DLL. Most likely due to errors in PLATFORM.DLL file format. System is in stable mode, no changes have been made to the BIOS, no re-boot is needed.

Stage 2 - Failure during platform dependent initialization. System is unstable and re-boot is needed. No changes have been made to the BIOS.

Stage 3 - Failure during programming of the flash memory. System is unstable and the BIOS flash memory is corrupted. System must be restarted with Crisis Diskette.

The program will also perform error detection at each of the steps. Unless explicitly disabled by the PRODUCTION option, as the program progresses it will report on the screen and debug port either the step number of the step currently in progress, or one of the error codes. Code numbers for errors and program steps are further defined in Appendix B2:

When an error is detected before any changes are made to the flash memory part, then program will attempt to notify the user and then will exit PHLASHNT with proper error messages. Once the flash memory has been modified, errors will cause the program to halt.

2.4 Device Dependent Modules

For each type of flash memory supported, there will be a part specific module to perform the following:

1) Identify the part and return the manufacturer ID and the part ID

2) Zero a range of flash memory (set all bits to 0)

3) Erase a range of flash memory (set all bits to 1)

4) Program a range of flash memory

2.4.1 Autodetection

In this module will be the code necessary to read the Manufacturer ID and part ID from the flash memory part. If IDs cannot be determined, zero is returned. The built-in auto-detect module will not be used when the AutoSense() function is provided in the PLATFORM.DLL file; the provided AutoSense() function will be used instead.

2.4.2 Zero

There are several part types which require that the flash memory is set to zero before it can be programmed. In this module will be the code necessary to set memory range to zeros.

2.4.3 Erase

Most flash memory parts require that the flash memory is set to all ones before the part can be programmed. These parts often allow erasure of a full block of flash memory with a single write operation. In this module will be the code necessary to set memory range to ones.

When block descriptors are defined in the PLATFORM.DLL file, descriptors must be set up so that there is at least one descriptor for each "erasable" block in the flash memory. For example in Intel 28F004 flash memory there is one 16kByte BOOT block, two 8kByte PARAMETER blocks, one 96kByte MAIN block and three 128kByte EXTENDED blocks. Each of the seven blocks can be erased with a single write and there must be at least one descriptor for each of the seven blocks.

2.4.4 Program

In this module will be the code necessary to read the data bytes form the BIOS ROM image and program these into the flash part.

2.5 Supported Devices

Initial set of flash memory devices supported by PHLASHNT.EXE will include the parts listed in the table below. For each part type a manufacturer and part ID and part description is listed. As new parts become available it may be necessary to add additional modules to PHLASHNT.EXE so that a new type of flashing algorithm is provided (new AutoDetect(), Zero(), Erase() and Program() functions). If it is possible for the new part to use one of the existing algorithms and only the Manufacturer or Part ID changes, then this can be indicated in the PLATFORM.DLL file and no modification of PHLASHNT.EXE is needed (see section on PartTypes for more detail).

Type	Mfg ID	Part ID	Description
2	0x01	0xA1	AMD 28F256
2	0x01	0x25	AMD 28F512
1	0x01	0xA7	AMD 28F010
1	0x01	0xA2	AMD 28F010A
2	0x01	0x2A	AMD 28F020

2	0x01	0x29	AMD 28F020A
2	0x01	0x20	AMD 29F010
2	0x01	0xA4	AMD 29F040
2	0x01	0x51	AMD 29F200T
10	0x01	0xB0	AMD 29F002T
10	0x01	0x34	AMD 29F002B
10	0x01	0xDC	AMD 29F002BXT
10	0x01	0x5D	AMD 29F002BXB
3	0x1F	0xD5	ATMEL 29C010
8	0x1F	0xDA	ATMEL 29C020
3	0x1F	0x35	ATMEL 29LV010
1	0x1C	0xD0	Mitsubishi 28F101
1	0x89	0xB9	Intel 28F256
1	0x89	0xB8	Intel 28F512
1	0x89	0xB4	Intel 28F010
1	0x89	0xBD	Intel 28F020
4	0x89	0x94	Intel 28F001 BX-T
4	0x89	0x95	Intel 28F001 BX-B
4	0x89	0x7C	Intel 28F002 BX-T
4	0x89	0x7D	Intel 28F002 BX-B
4	0x89	0x74	Intel 28F200 BX-T
4	0x89	0x75	Intel 28F200 BX-B
4	0x89	0x78	Intel 28F004 BX-T
4	0x89	0x79	Intel 28F004 BX-B
4	0x89	0x70	Intel 28F400 BX-T
4	0x89	0x71	Intel 28F400 BX-B
5	0xBF	0x07	SST 29EE010/29LE010
12	0xBF	0x10	SST 29EE020
5	0xBF	0x5D	SST 29EE512
6	0xBF	0x04	SST 28SF040
1	0x20	0x07	ST M28F101
7	0xC2	0x11	MX 28F1000
11	0xC2	0x2A	MX 28F2000

3.0 PLATFORM.DLL Detail

This module contains all platform dependent code and parameters needed to program a flash device on a particular platform.

3.1 File Format

PLATFORM.DLL is a Windows DLL that is produced by compiling PLATFORM.CPP (using Microsoft Visual C++ 4.2 or later). It contains specific platform data and executable code. A sample source code of the PLATFORM.CPP file is included in Appendix B3.

File version for PLATFORM.DLL will start with version "NT 1.00". Versions are specified in the szVersion variable contained in PLATFORM.DLL.

3.2 File Header Format

The PLATFORM.DLL file will have the format described below:

```
// ------------------------------
// Global Variable Declarations
// ------------------------------

DWORD    dwFileSize        // ROM image file size
BYTE     bManufactID       // Manufacturer of flash device
BYTE     bPartID           // Part ID of flash device
DWORD    dwFlags           // Option flags
DWORD    dwImageBuf        // Linear address of image buffer
DWORD    dwMfgIDAddr       // Linear address of mfg ID
DWORD    dwPartIDAddr      // Linear address of part ID
BYTE     bRetryCount       // Count for /Rn option (default = 0)
char     szVersion[]       // PLATFORM.DLL version
char     szROMFileName[]   // Name of BIOS image file
DWORD    dwDLLFuncDefine   // Indicates what functions are defined
BYTE     bblockTableSize   // number of blocks in blockTable
BLOCK_DESCRIPTOR blockTable[]
BYTE     bpartTypesSize    // number of flash parts to add
DEVICETABLE partTypes[]
```

dwFileSize Number of bytes in the BIOS.ROM file.

bManufactID Manufacturer ID

bPartID Part ID

dwFlags Option flags. Must be combination of the following values:

FLAG_AUTOSENSEOFF	Don't read ID from the part
FLAG_BACKUP	Backup system BIOS ROM
FLAG_NEWBIOSONLY	Don't flash if 64k at F000:0 same
FLAG_PRODUCTION	Max speed (sound & video off)

FLAG_SILENT	Do not generate any sounds
FLAG_VERIFY	Verify each block after flash
FLAG_PLATFORMCMD	PLATFORM option str present
FLAG_BIOSPARTNUM	Flash only same BIOS part number
FLAG_CHECKSUM	Checksum BIOS.ROM
FLAG_CMOS	Clear CMOS checksum
FLAG_IMAGESIZE	Verify image size matches flash part

dwImageBuf Address for BIOS.ROM image buffer in extended memory. This field determines the linear address of a buffer where the image will be read into.

This area is also used when the SAVE option is specified. Any blocks to be saved will be using the address range starting at the address dwImageBuffer + dwFileSize.

dwMfgIDAddr
dwPartIDAddr These two optional fields contain the linear addresses for the flash memory ID bytes. When these fields are zero, the default addresses of E0000h and E0001h will be used.

bRetryCount Number of times to retry if flashing fails.

szVersion Version of PLATFORM.DLL

szROMFileName Reserved must be the string "BIOS.ROM". This field is used to identify and verify format of the PLATFORM.DLL file.

dwDLLFuncDefine Indicates what platform-specific functions are defined in PLATFORM.DLL

bblockTableSize `number of blocks described in blockTable`

dwBlockTable Flash parts are programmed one contiguous block at a time. Each block to be programmed must have a corresponding block descriptor.

bpartTypesSize `number of flash parts to add`

dwPartTypes Optional table of supported flash parts. Each entry in the table has the following format:

```
typedef struct
{
      BYTE  cMfgID;           // Manufacturer ID
      BYTE  cPartID;          // Part ID
      WORD  wFlashType;       // Flash algorithm type
      char  szPartName[28];   // Optional description
} DEVICETABLE;
```

The device table is terminated by a descriptor with cMfgID, cPartID and wFlashType set to zero.

Many platforms allow the same BIOS.ROM image to be used with several different flash parts. This table is used when a new part becomes available, the part is not among the parts currently supported by PHLASHNT.EXE and the part uses the same flashing algorithms as one of the supported parts.

procEnable	enable flashing proc
procDisable	disable flashing proc
procBegin	begin flashing proc
procEnd	end flashing proc
procGetBlock	get next BIOS block for backup proc
procCmdLine	process custom command line options proc
procSense	custom autosense proc
procIsFlashable	custom OEM proc to determine if ok to proceed
procReboot	custom reboot proc
procCheckSum	custom checksum proc to be used if the BIOS ROM image checksum is not zero

A sample source code for PLATFORM.DLL file is shown in Appendix B3.

3.3 Block Table Format

Block table consists of a list of block descriptors. Each block descriptor in the block table is defined by the following structure:

```
typedef struct
        {
        DWORD dwBlockSize;        // Number of bytes in the block
        DWORD dwFileOffset;       // Offset within BIOS.ROM file
        DWORD dwLinearAddress;    // Linear 32-bit address of flash ROM
        BYTE  cMfgID;             // Manufacturer ID or zero
        BYTE  cPartID;            // Part ID or zero
        WORD  wBlockAttr;         // Block attributes
        } BLOCK_DESCRIPTOR;
```

The block table is terminated by a descriptor with all entries set to zero.

dwBlockSize	Block Size in bytes. Blocks must be contiguous.
dwFileOffset	Offset of this block within the BIOS.ROM file.
dwLinearAddress	Starting address of this block in 32-bit address space
cMfgID	Manufacturer ID or zero to auto-sense.
cPartID	Part ID or zero to auto-sense.
wBlockAttr	Determines actions to be taken for this block. Must be a combination of the following flags:

ATTR_ZERO	Block must be zeroed before programming
ATTR_ERASE	Block must be erased before programming
ATTR_SAVE	Save content of this block before prog.
ATTR_PROG	Program this block
ATTR_RESTORE	Restore content of this block after prog.

Only one of ATTR_SAVE, ATTR_PROG, ATTR_RESTORE can be used at a time. If no attribute is specified, then PHLASHNT.EXE leaves the block unchanged. However, even if all of these are omitted, procedures BeginFlash() and EndFlash() are still called. BeginFlash() and EndFlash() can be used when two blocks are in different flash devices, or when a boot block requires additional functions to enable the block for write and to disable it before next block is programmed. BeginFlash() can also be used for conditional block processing. If BeginFlash() returns nonzero, the current block is not processed.

Each ATTR_SAVE block must be followed by an ATTR_RESTORE block before another ATTR_SAVE block can be used.

Note that for a given flash memory range there may be several block descriptors. For example to preserve a 16k flash memory Boot Block in 64k erasable flash memory block, three block descriptors would be used. First descriptor to save the 16k boot block, second to erase and program 64k and third to restore the boot block.

To reduce the time required for flashing, it is recommended that the ATTR_ZERO flag is not used, because this will avoid the zeroing step and cut the flashing time in half. Only few of the older flash memory types suggested that part is zeroed before it is re-programmed. Most parts do not require this operation.

3.3.3 Multiple Flash Blocks

The block table will be used to support multiple device flashing and multiple blocks within each device. For such platforms the ROM image file must contain the images for all flash parts to be programmed and the Block Table must contain proper offsets and lengths for each block of data to be flashed.

To properly configure the platform before and after each flash block, PHLASHNT.EXE will call function BeginFlash(Block_Index) to allow PLATFORM.DLL to perform any such set-up. It is the responsibility of BeginFlash() & EndFlash() functions to perform any initialization or termination between blocks as needed.

3.3.4 Processing Boot Blocks and ESCD storage

To program a memory range in the flash memory, there may have to be several different block descriptors, for the same memory range. This may be needed to preserve boot block or ESCD storage within a single 'erase' memory range.

Many flash parts are erased by a small number of write operations, one for each memory block. For example Intel 28F400 flash memory has seven blocks (one 16k boot block, two 8k parameter blocks, one 64k main block and three 128k extended blocks). This part can be erased with only seven write operations.

For other parts, erase function may erase 64k of flash memory at a time, regardless of division of this range into boot and parameter blocks. In such cases it is important that there are three block descriptors for such a 64k range of memory. The first block descriptor in the table is used to save boot block, second block descriptor to erase and program the parameter blocks and the third descriptor to restore the boot block in this range.

Some parts require that a additional platform dependent actions need to be taken before a boot block can be programmed. For example Intel parts require that VHH voltage, in addition to VPP voltage, is properly set. In such cases block descriptor must also have such a functions in **BeginFlash()** and **EndFlash()** procedures (called before and after each block).

3.3.5 Block Table Examples

The code in the following examples would be placed in the "blockTable" section of PLATFORM.CPP.

Erase a 4KB block at FC000.

```
DD      4 * 1024                ; 4KB
DD      0                       ; File offset
DD      000FC000h               ; Linear address of this block
DB      0                       ; Mfg ID  (0 = default)
DB      0                       ; Part ID (0 = default)
DW      ATTR_ERASE              ; Action flags
```

Zero, then program a 128KB block at E0000 on the specified part.

```
DD      128 * 1024              ; 128KB
DD      0                       ; File offset
DD      000E0000h               ; Linear address of this block
DB      01h                     ; Mfg ID  (0 = default)
DB      20h                     ; Part ID (0 = default)
DW      ATTR_ZERO OR ATTR_PROG  ; Action flags
```

3.4 PLATFORM.DLL Functions

Currently supported functions are:

The following functions contained in PLATFORM.DLL are accessed by PHLASHNT.EXE to implement platform-dependent functionality:

EnableFlash
DisableFlash
BeginFlash
EndFlash
GetBlock
CmdLine
AutoSense
IsFlashable
Reboot
CheckSum

The following functions allow PHLASHNT.EXE to access global variables and data structures contained in PLATFORM.DLL:

GetBIOSFileSize
GetManufactID
GetPartID
GetFlags
GetImageBuf
GetMfgIDAddr
GetPartIDAddr
GetRetryCount
GetblockTableSize
GetpartTypesSize
GetBlockTable
GetpartTypes
GetDLLVersion
GetROMFileName
GetDLLFuncDefine

3.4.1 Function EnableFlash()

Entry: None
Returns: Error code (or zero)

This function must be present in the PLATFORM.DLL. It is called before any attempt to access the flash memory. Actions typically performed by this function include:

Map flash device into memory
Disable Cache, Shadowing and Power management
Flush Cache
Disable PCI bridge
Enable ROM for write (VPP on)

Most platforms require that a jumper is changed before the part can be enabled for flashing. EnableFlash() procedure must verify that this jumper has been removed and return an error code if it was determined that jumper settings are incorrect. See Appendix B for error codes.

3.4.2 Function DisableFlash()

Entry: None
Returns: Error code (or zero)

This optional function is called after the last block has been programmed (or error was detected). It will be called immediately before PHLASHNT.EXE exits (typically as a last call before re-boot). It should perform all functions necessary to be able to cleanly re-

Action typically performed by this function include:

Disable ROM Write (VPP off)

3.4.3 Function BeginFlash(DWORD Block_Index)

Entry: Index of the block about to be programmed (or zero if table not used)
Exit: None
Returns: Error code (or zero)

This optional function is called by PHLASHNT.EXE immediately before the flash block is processed. It will be called for each block (found in the block table).

Actions typically performed by this function include:

Save BOOT block before it is erased
Switch from one device to another (on platforms with multiple devices)
Enable VHH for boot block re-program
Determine if the current block should be processed

If BeginFlash() returns nonzero, the current block will not be processed.

3.4.4 Function EndFlash(DWORD Block_Index)

Entry: Index of the block just programmed (or zero of table not used)
Returns: Error code (or zero)

This optional function is called by PHLASHNT.EXE immediately after the flash block was processed. It will be called for each block (found in the block table).

Actions typically performed by this function include:

Restore BOOT block saved by BeginFlash() function
Clean-up between programming two different devices
Disable VHH if boot block was just programmed

3.4.5 Function GetBlock(DWORD Index, DWORD Buffer_Address)

Entry: Index of the block to be copied and linear address of a 64k buffer
Exit: Buffer is filled with the next block of existing BIOS ROM image
Returns: Negative error code, zero, or positive block index

This optional function is called by PHLASHNT.EXE whenever the /BACKUP flag is specified. GetBlock() is used to assist when saving the existing content of the flash memory before the flash memory is changed. Because many BIOS images are de-compressed into shadow RAM, it is not always possible for PHLASHNT.EXE to access all of the BIOS ROM image without platform dependent system setup. Function GetBlock() is necessary to allow for platform dependent accessing of the existing BIOS ROM image. BIOS ROM image is saved by PHLASHNT.EXE using the following steps:

1) Call GetBlock(Index, Buffer) with Index set to 0 and the 64k buffer, pointed to by the parameter Buffer, filled with a pre-defined pattern. If the pattern in the buffer is not changed, program exits with error. If the pattern is changed, then the buffer is saved as the first 64k block in the BIOS.BAK file, then program proceed to next step.

2) Call GetBlock(Index, Buffer) with Index set to the value returned by the previous call to GetBlock(), save the 64k buffer into BIOS.BAK and repeat this until the value returned by GetBlock() is a non-positive number.

3) If the last value returned by GetBlock() is zero, then proceed with flashing of the memory. If the last value returned is negative error code, report the error, delete BIOS.BAK and exit.

It is the responsibility of the GetBlock() implementation to ensure that the platform is in a proper state to allow the GetBlock() to copy BIOS ROM image into the buffer and that the platform is restored to the original mode before GetBlock() returns control to PHLASHNT.EXE. In particular, GetBlock() is called before a call is made to EnableFlash(). The buffer pointer passed to GetBlock() is always in the real memory range below 640k, to allow direct transfer to disk.

3.4.6 Function CmdLine(char far *szOptions)

Entry: Pointer to a string with the platform specific command line options
Returns: Error code (or zero)

This optional function is called by PHLASHNT.EXE immediately after the PLATFORM.DLL was read in. It is passed the address of the string containing all the platform specific command line parameters (specified after the equal sign with /PLATFORM="..." command line option). The string includes the leading and trailing double quotes, if any.

3.4.7 Function AutoSense()

Entry: Manufacturer and device IDs retrieved from PLATFORM.INI header
Returns: New ID retrieved from the flash part (or zero)

This function is called by PHLASHNT.EXE immediately after EnableFlash() function in the PLATFORM.DLL was called. The AutoSense() function enables auto detection of memory flash parts when "non-standard" memory organization is used for the flash memory. For example when two separate parts are used for even and odd BIOS addresses (in which case the conventional auto detect mechanism will fail), this function can be used to obtain and verify ID for each of the parts.

IDs are one byte long and are packed into a DWORD, with manufacturer ID in BYTE 0 and device ID in BYTE 1.

3.4.8 Function IsFlashable(char far *szErrorMsg)

Entry: Pointer to string to contain returned error message.
Exit: szErrorMsg containing error message string.
Returns: Error code (or zero)

This optional function is called before EnableFlash() to determine if it is ok to proceed. If the function returns a nonzero error code, the string szErrorMsg is displayed and the program terminates. Up to 254 bytes plus a terminating NULL may be returned in szErrorMsg.

An example of how this might be used is: for the same platform, an OEM sells a system with and without Plug and Play capabilities. The IsFlashable() function can determine if the system currently has Plug and Play and will not flash a Plug and Play BIOS on a platform that doesn't already have it.

3.4.9 Function Reboot()

Entry: None.
Returns: None.

This optional function is called after programming is complete to reset the system. If provided, this is called instead of PHLASHNT's own reboot code.

3.4.10 Function CheckSum()

Entry: None.
Returns: Error code (or zero)

This optional function is called before programming to determine if the checksum of the BIOS ROM image is correct. Normally, the BIOS ROM image checksum for a NuBIOS image is zero. This routine may be used to provide an alternative checksum verification method when it is known that the ROM image checksum will not be zero. If this function returns non-zero, PHLASHNT will terminate.

3.4.11 Function GetBIOSFileSize()

Entry: None.
Returns: Size of BIOS image

This function returns the contents of the global variable dwFileSize from PLATFORM.DLL.

3.4.12 Function GetManufactID()

Entry: None.
Returns: Manufacturer ID

This function returns the contents of the global variable bManufactID from PLATFORM.DLL.

3.4.13 Function GetPartID()

Entry: None.
Returns: Part ID

This function returns the contents of the global variable bPartID from PLATFORM.DLL.

3.4.14 Function GetFlags()

Entry: None.
Returns: option flags

This function returns the contents of the global variable dwFlags from PLATFORM.DLL.

3.4.15 Function GetImageBuf()

Entry: None.
Returns: Location of Image Bufffer

This function returns the contents of the global variable dwImageBuf from PLATFORM.DLL.

3.4.16 Function GetMfgIDAddr()

Entry: None.
Returns: Address where Manufacturer ID is located

This function returns the contents of the global variable dwMfgIDAddr from PLATFORM.DLL.

3.4.17 Function GetPartIDAddr()

Entry: None.
Returns: Address where Part ID is located

This function returns the contents of the global variable dwPartIDAddr from PLATFORM.DLL.

3.4.18 Function GetRetryCount()

Entry: None.
Returns: Number of times to retry flashing if failure occurs

This function returns the contents of the global variable bRetryCount from PLATFORM.DLL.

3.4.19 Function GetblockTableSize()

Entry: None.
Returns: Number of blocks in blockTable

This function returns the contents of the global variable bblockTableSize from PLATFORM.DLL.

3.4.20 Function GetpartTypesSize()

Entry: None.
Returns: Number of additional flash parts that PLATFORM.DLL will describe

This function returns the contents of the global variable bpartTypesSize from PLATFORM.DLL.

3.4.21 Function GetBlockTable()

Entry: None.
Returns: address of blockTable

This function returns the address of the global structure blockTable from PLATFORM.DLL.

3.4.22 Function GetpartTypes()

Entry: None.
Returns: address of partTypes

This function returns the address of the global structure partTypes from PLATFORM.DLL.

3.4.23 Function GetDLLVersion()

Entry: None.
Returns: Version of PLATFORM.DLL

This function returns the contents of the global variable szVersion from PLATFORM.DLL.

3.4.24 Function GetROMFileName()

Entry: None.
Returns: Name of BIOS ROM file

This function returns the contents of the global variable szROMFileName from PLATFORM.DLL.

3.4.25 Function GetDLLFuncDefine()

Entry: None.
Returns: Indicates what platform-dependent functions are defined in PLATFORM.DLL

This function returns the contents of the global variable dwDLLFuncDefine from PLATFORM.DLL.

4.0 BIOS.ROM Detail

The ROM image file size must be large enough to contain all blocks in the flash device(s) to be programmed. Any required post-processing of the BIOS image, such as boot block swapping or data compression, must be already incorporated into the ROM image file.

5.0 General Implementation Guidelines

Programs will be developed using Microsoft Visual C++, V4.2 or later.

Because it is expected that this program will evolve as new devices become available and because code size is not critical, the style of coding should be such that code readability and clarity should have higher priority than executable code compactness.

However, because the program may also be used in production environments, it should be structured as to allow shortest possible time for part flashing. In particular, it should have a mode where non-critical user interface notifications can be disabled; and whenever possible, time consuming flashing functions should be written in optimized assembly language.

Appendix B1 - Future enhancements

Completion Codes with Keyboard LEDs

If the program fails to complete any of the three major stages of the flashing process, program will use keyboard LEDs to inform user at which stage the program failed. At the start of the program CAPS_LOCK, NUM_LOCK and SCROLL_LOCK LEDs on the keyboard will be turned on. The failure at each of the three stages is indicated as follows:

Keyboard LEDs ON	Description
CAPS, NUM, SCROLL	Before reading platform.dll
CAPS, NUM	Before platform init
NUM	After platform init
none	Successful completion

Appendix B2-PHLASHNT.H Completion and error codes

FFh	Memory alloc for BIOS.BAK buffer failed
FEh	BIOS.BAK already exists (rename or delete it)
FDh	File Create failed on BIOS.BAK
FCh	File Write failed on BIOS.BAK
FBh	File Close failed on BIOS.BAK
FAh	BIOS backup not supported in PLATFORM.DLL
F9h	File Open failed on PLATFORM.DLL
F8h	File Read failed on PLATFORM.DLL
F7h	File Close failed on PLATFORM.DLL
F6h	Failed to locate signature bytes in PLATFORM.DLL
F5h	Unsupported PLATFORM.DLL file version
F2h	Device table in PLATFORM.DLL has too many entries
F1h	Device table in PLATFORM.DLL has unsupported flash type
F0h	Combined SAVE or RESTORE attributes in PLATFORM.DLL
EFh	SAVE block without matching RESTORE block in PLATFORM.DLL
ECh	Part ID not found in table of supported parts
EBh	PLATFORM.DLL found errors in command line parameters
EAh	Alloc for BIOS ROM image failed
E9h	Open failed on BIOS ROM image file
E8h	Read failed on BIOS ROM image file
E6h	File Close failed on BIOS.ROM
E4h	Attempt to read flash memory ID failed
E3h	PLATFORM.DLL failed to return flash memory ID
E2h	Could not find BCPSYS block in BIOS ROM file image
E1h	File does not contain the same BIOS part number
E0h	File does not contain different version of BIOS ROM image
DFh	Data written to flash does not match BIOS ROM image
DEh	Write to flash memory failed
DDh	Erase flash memory block failed
DCh	VPP is not at expected level
DBh	Erase sequence failed
DAh	New DMI string is too large
D9h	The BIOS ROM file may not be used with this system
D8h	Alloc for DMI OEM string failed
D7h	No space for the specified DMI OEM string in the BIOS ROM
D6h	DMI OEM strings not supported (BCPDMI 0.1+ required)
D5h	Could not find BCPDMI block in BIOS ROM file image
D3h	BIOS ROM file may be corrupt (checksum not zero)
D2h	BIOS ROM file size doesn't match flash part size
D1h	DMI system and chassis strings require BCPDMI 2.1+

Appendix B3-PLATFORM.DLL Sample source code

PLATFORM.CPP

```
/*
 *
 * TITLE   PLATFORM.CPP  - 32-Bit DLL to provide platform dependent functionality
 *                         to PHLASHNT.EXE
 *
 * Copyright (c) 1997 by Phoenix Technologies, Ltd., All Rights Reserved.
 * Phoenix Technologies Ltd. CONFIDENTIAL.
 *
 *************************************************************************
 *
 * Filename:           PLATFORM.CPP
 *
 * Project:            PHLASHNT.EXE
 *
 * Functional Block:
 *
 * Comments:   The variable dwDLLFuncDefine is used in PHLASHNT to determine
 *             what platform specific functions in PLATFORM.DLL have been
 *             defined.  Listed below are the possible values of
 *             dwDLLFuncDefine and what functions they indicate are defined
 *             in this module:
 *
 *                     DLL_ENABLEFLASH       EnableFlash
 *                     DLL_DISABLEFLASH      DisableFlash
 *                     DLL_BEGINFLASH        BeginFlash
 *                     DLL_ENDFLASH          EndFlash
 *                     DLL_GETBLOCK          GetBlock
 *                     DLL_CMDLINE           CmdLine
 *                     DLL_AUTOSENSE         AutoSense
 *                     DLL_ISFLASHABLE       IsFlashable
 *                     DLL_REBOOT            Reboot
 *                     DLL_CHECKSUM          CheckSum
 *
 * Contents:
 *
 *************************************************************************
 *
 * Version Control Information:
 *
 * $Log:   K:/nb/archive/nutools/phlash.nt/stage2/drivers/platform.cpv  $
 *
 *        Rev 1.0
 *     Initial revision.
 *
 *************************************************************************/

#include "stdafx.h"
#include <afxdllx.h>

#include <stdio.h>
#include "D:\NUTOOLS\PHLASH.NT\STAGE2\phlashnt.h"

#define PLATFORM_CPP


        // ---------------------------------
        // Global Variable Declarations
        // ---------------------------------

DWORD   dwFileSize      = 0x00040000;  // ROM image file size
BYTE    bManufactID     = 0x89;        // Manufacturer of flash device
BYTE    bPartID         = 0xBD;        // Part ID of flash device

                                       // Option flags
DWORD   dwFlags = FLAG_BIOSPARTNUM | FLAG_CHECKSUM | FLAG_IMAGESIZE;
DWORD   dwImageBuf      = 0x00200000;  // Linear address of image buffer
DWORD   dwMfgIDAddr     = 0xFFFE0000;  // Linear address of mfg ID
```

```
DWORD   dwPartIDAddr    = 0xFFFE0001;  // Linear address of part ID
BYTE    bRetryCount     = 0;           // Count for /Rn option (default = 0)
char    szVersion[]     = "NT 1.00";   // PLATFORM.DLL version
char    szROMFileName[] = "BIOS.ROM";  // Name of BIOS image file

            // Indicates what functions are defined in PLATFORM.DLL
DWORD   dwDLLFuncDefine =       DLL_ENABLEFLASH
                                DLL_DISABLEFLASH
                                DLL_BEGINFLASH
                                DLL_ENDFLASH
                                DLL_GETBLOCK
                                DLL_CMDLINE
                                DLL_AUTOSENSE
                                DLL_ISFLASHABLE
                                DLL_REBOOT
                                DLL_CHECKSUM ;


                                // ------------------------
                                // define blockTable
                                // ------------------------
BYTE    bblockTableSize         = 5;            // number of blocks in blockTable
BLOCK_DESCRIPTOR blockTable[] =
{
        {
        128 * 1024,     // 128KB (Low)
        0,              // File offset
        0xFFFE0000,     // Linear address of flash ROM
        0,              // MfgID  (same as default)
        0,              // PartID (same as default)
        ATTR_ZERO
        },


        {
        128 * 1024,     // 128KB (High)
        0x00020000,     // File offset
        0xFFFE0000,     // Linear address of flash ROM
        0,              // MfgID  (same as default)
        0,              // PartID (same as default)
        ATTR_ZERO
        },

        {
        128 * 1024,     // 128KB (Low)
        0,              // File offset
        0xFFFE0000,     // Linear address of flash ROM
        0,              // MfgID  (same as default)
        0,              // PartID (same as default)
        ATTR_ERASE
        },

        {
        128 * 1024,     // 128KB (Low)
        0,              // File offset
        0xFFFE0000,     // Linear address of flash ROM
        0,              // MfgID  (same as default)
        0,              // PartID (same as default)
        ATTR_PROG
        },

        {
        128 * 1024,     // 128KB (High)
        0x00020000,     // File offset
        0xFFFE0000,     // Linear address of flash ROM
        0,              // MfgID  (same as default)
        0,              // PartID (same as default)
        ATTR_PROG
        },

        {               // Need to terminate blockTable with
        0,              // a block set to 0.
        0,
        0,
        0,
        0,
```

```
        0
        }
};

                                // ==========================
                                // define partTypes
                                // ==========================
BYTE    bpartTypesSize = 1;             // number of flash parts on platform
                                        // Set to 0 if partTypes is not used!
DEVICETABLE partTypes[] =
{
        {
        0x89,                   // Manufacturer ID of flash device
        0xBD,                   // Device ID of flash device
        1,                      // Flashing algorithm type
        0,                      // IGNORE wPartSize
        "Intel 28F020"          // Name of flash device
        },

        {                       // Need to terminate partTypes with
        0,                      // a block set to 0.
        0,
        0,
        0,
        ""
        }
};
```

```
HINSTANCE       hKernel32 = 0;      // NOTE:  Do NOT change this line of code.

 /*************************************************************************
 *                       Functions
 *************************************************************************/


 /*************************************************************************
 *
 * Function Name: DllMain
 *
 * Description:
 * Parameters:
 * Returns:
 *
 * Notes:       DO NOT MODIFY THIS FUNCTION
 *
 *************************************************************************/
extern "C" int APIENTRY
DllMain(HINSTANCE hInstance, DWORD dwReason, LPVOID lpReserved)
{
        if (dwReason == DLL_PROCESS_ATTACH)
        {
                TRACE0("PLATFORM.DLL Begin!\n");
                hKernel32 = LoadLibrary( "Kernel32.Dll" ); // Get Handle to Kernel32.Dll
        }
        else if (dwReason == DLL_PROCESS_DETACH)
        {
                TRACE0("PLATFORM.DLL Terminating!\n");
                if (hKernel32)
                        FreeLibrary( hKernel32 );
        }
        return 1;   // ok

} // DllMain()

 /*************************************************************************
 *
 * Function Name: EnableFlash
 *
 * Description:
 * Parameters:
```

```
 * Returns:
 *
 * Notes:
 *
 ************************************************************************/
DLL_FUNC_TYPE void EnableFlash()
{

  printf("EnableFlash\n");  //  (stub)

}
 /************************************************************************
 *
 * Function Name: DisableFlash
 *
 * Description:
 * Parameters:
 * Returns:
 *
 * Notes:
 *
 ************************************************************************/
DLL_FUNC_TYPE void DisableFlash()
{

  printf("DisableFlash\n");  //  (stub)

}
 /************************************************************************
 *
 * Function Name: BeginFlash
 *
 * Description:
 * Parameters:
 * Returns:
 *
 * Notes:
 *
 ************************************************************************/
DLL_FUNC_TYPE short BeginFlash( USHORT blockNumber )
{

  return 0;     // (stub)

}

 /************************************************************************
 *
 * Function Name: EndFlash
 *
 * Description:
 * Parameters:
 * Returns:
 *
 * Notes:
 *
 ************************************************************************/
DLL_FUNC_TYPE short EndFlash( USHORT blockNumber )
{

  return 0;     //  (stub)

}
 /************************************************************************
 *
 * Function Name: GetBlock
 *
 * Description:
 * Parameters:
 * Returns:
 *
 * Notes:
 *
 ************************************************************************/
DLL_FUNC_TYPE SHORT GetBlock( ULONG dwIndex, PULONG dwDst )
{
```

```
    printf("GetBlock:  dwIndex,dwDat passed in=%d,%d\n", dwIndex, dwDat );  //  (stub)
    return 0;

}
 /*****************************************************************************
 *
 * Function Name: CmdLine
 *
 * Description:
 * Parameters:
 * Returns:
 *
 * Notes:
 *
 *****************************************************************************/
DLL_FUNC_TYPE USHORT CmdLine( char *PlatformString )
{

  printf("CmdLine\n");  //  (stub)
  return( 0 );          //  (stub)

}
 /*****************************************************************************
 *
 * Function Name: AutoSense
 *
 * Description:
 * Parameters:
 * Returns:
 *
 * Notes:
 *              The manufacturer ID is in BYTE 0 and device ID in BYTE 1
 *
 *****************************************************************************/
DLL_FUNC_TYPE DWORD AutoSense( DWORD PartMfgID )
{
  return( 0x0000 );    //  (stub)

}
 /*****************************************************************************
 *
 * Function Name: IsFlashable
 *
 * Description:
 * Parameters:
 * Returns:
 *
 * Notes:
 *
 *****************************************************************************/
DLL_FUNC_TYPE USHORT IsFlashable()
{

  printf("IsFlashable\n" );  //  (stub)
  return 0;

}
 /*****************************************************************************
 *
 * Function Name: Reboot
 *
 * Description:
 * Parameters:
 * Returns:
 *
 * Notes:
 *
 *****************************************************************************/
DLL_FUNC_TYPE void Reboot()
{

  printf("Reboot\n");  //  (stub)

}
 /*****************************************************************************
```

```
 *
 * Function Name: CheckSum
 *
 * Description:
 * Parameters:
 * Returns:
 *
 * Notes:
 *
 ************************************************************************/
DLL_FUNC_TYPE void CheckSum()
{

  printf("CheckSum\n");  //  (stub)

}


// ------------------------------------------------------------------------
//              NOTE:  DO NOT MODIFY THE FUNCTIONS LISTED BELOW.
// ------------------------------------------------------------------------

DLL_FUNC_TYPE DWORD    GetBIOSFileSize(void)    { return(dwFileSize); }
DLL_FUNC_TYPE BYTE       GetManufactID(void)      { return(bManufactID); }
DLL_FUNC_TYPE BYTE       GetPartID(void)          { return(bPartID); }
DLL_FUNC_TYPE DWORD      GetFlags(void)           { return(dwFlags); }
DLL_FUNC_TYPE DWORD      GetImageBuf(void)        { return(dwImageBuf); }
DLL_FUNC_TYPE DWORD      GetMfgIDAddr(void)       { return(dwMfgIDAddr); }
DLL_FUNC_TYPE DWORD      GetPartIDAddr(void)      { return(dwPartIDAddr); }
DLL_FUNC_TYPE BYTE       GetRetryCount(void)      { return(bRetryCount); }
DLL_FUNC_TYPE BYTE       GetblockTableSize(void)  { return(bblockTableSize); }
DLL_FUNC_TYPE BYTE       GetpartTypesSize(void)   { return(bpartTypesSize); }
DLL_FUNC_TYPE BLOCK_DESCRIPTOR  *GetBlockTable(void) { return(&blockTable[0]); }
DLL_FUNC_TYPE DEVICETABLE       *GetpartTypes(void)  { return(&partTypes[0]); }
DLL_FUNC_TYPE char     *GetDLLVersion(void)   { return(szVersion); }
DLL_FUNC_TYPE char     *GetROMfileName(void)  { return(szROMFileName); }
DLL_FUNC_TYPE DWORD    GetDLLFuncDefine(void) { return(dwDLLFuncDefine); }
```